

4-10-00

A

UTILITY PATENT APPLICATION TRANSMITTAL

(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.

ST9-99-179

Total Pages in this Submission

145

TO THE ASSISTANT COMMISSIONER FOR PATENTS

Box Patent Application

Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

CROSS-PLATFORM SUBSELECT METADATA EXTRACTION

and invented by:

RICHARD HENRY MANDEL, III

jc564 U.S. PTO
09/545592
04/07/00

If a **CONTINUATION APPLICATION**, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Enclosed are:

Application Elements

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 27 pages and including the following:
 - a. ☒ Descriptive Title of the Invention
 - b. ☐ Cross References to Related Applications (if applicable)
 - c. ☐ Statement Regarding Federally-sponsored Research/Development (if applicable)
 - d. ☐ Reference to Microfiche Appendix (if applicable)
 - e. ☒ Background of the Invention
 - f. ☒ Brief Summary of the Invention
 - g. ☒ Brief Description of the Drawings (if drawings filed)
 - h. ☒ Detailed Description
 - i. ☒ Claim(s) as Classified Below
 - j. ☒ Abstract of the Disclosure

UTILITY PATENT APPLICATION TRANSMITTAL
(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.
ST9-99-179

Total Pages in this Submission
145

Application Elements (Continued)

3. ☒ Drawing(s) *(when necessary as prescribed by 35 USC 113)*
- a. ☒ Formal Number of Sheets 6
- b. ☐ Informal Number of Sheets _____
4. ☒ Oath or Declaration
- a. ☒ Newly executed *(original or copy)* ☐ Unexecuted
- b. ☐ Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional application only)*
- c. ☒ With Power of Attorney ☐ Without Power of Attorney
- d. ☐ DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application,
see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. ☐ Incorporation By Reference *(usable if Box 4b is checked)*
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Computer Program in Microfiche *(Appendix)*
7. ☐ Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all must be included)*
- a. ☐ Paper Copy
- b. ☐ Computer Readable Copy *(identical to computer copy)*
- c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

Accompanying Application Parts

8. ☒ Assignment Papers *(cover sheet & document(s))*
9. ☐ 37 CFR 3.73(B) Statement *(when there is an assignee)*
10. ☐ English Translation Document *(if applicable)*
11. ☒ Information Disclosure Statement/PTO-1449 ☒ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgment postcard
14. ☒ Certificate of Mailing
- ☐ First Class ☒ Express Mail *(Specify Label No.):* EL 243 341 033 US

UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.
ST9-99-179

Total Pages in this Submission
145

Accompanying Application Parts (Continued)

15. ☐ Certified Copy of Priority Document(s) (if foreign priority is claimed)
16. ☐ Additional Enclosures (please identify below):

--

Fee Calculation and Transmittal


CLAIMS AS FILED

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	33	- 20 =	13	x \$18.00	\$234.00
Indep. Claims	3	- 3 =	0	x \$78.00	\$0.00
Multiple Dependent Claims (check if applicable) <input type="checkbox"/>					\$0.00
BASIC FEE					\$690.00
OTHER FEE (specify purpose)					\$0.00
TOTAL FILING FEE					\$924.00

- ☐ A check in the amount of _____ to cover the filing fee is enclosed.
- ☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. **09-0460** as described below. A duplicate copy of this sheet is enclosed.
- ☒ Charge the amount of **\$924.00** as filing fee.
- ☒ Credit any overpayment.
- ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
- ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

Dated: April 7, 2000

cc:


Signature
Janaki Komanduri, Esq.
Reg. No. 40,684
Pretty, Schroeder & Poplawski
444 S. Flower Street, 19th Floor
Los Angeles, California 90071
Tel: (213) 622-7700
Fax: (213) 489-4210

UTILITY APPLICATION
OF
Richard Henry Mandel, III
FOR
UNITED STATES LETTERS PATENT
ON
CROSS-PLATFORM SUBSELECT METADATA EXTRACTION

Docket No.: ST9-99-179 (IBMST 044276)

Drawings: 6

Attorneys
PRETTY, SCHROEDER & POPLAWSKI
444 South Flower Street, 19th Floor
Los Angeles, California 90071
Ofc: 213/622-7700
Fax: 213/489-4210

CERTIFICATE OF MAILING BY "EXPRESS MAIL"

"EXPRESS MAIL" MAILING LABEL NUMBER EL243341033US

DATE OF DEPOSIT April 7, 2000

I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES
POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 CFR-10 ON THE
DATE INDICATED ABOVE AND IS ADDRESSED TO BOX PATENT APPLICATION, THE ASSISTANT
COMMISSIONER FOR PATENTS, WASHINGTON, D C , 20231.

Joanne H. Housen

(TYPED OR PRINTED NAME OR PERSON MAILING PAPER OR FEE)

Joanne H. Housen
(SIGNATURE OF PERSON MAILING PAPER OR FEE)

[illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible]

or query metadata information. Metadata information comprises information about other information. For example, metadata provides type information about columns in a table.

SQLJ is a set of standards used to define how the Java™ programming language can be used with the Structured Query Language (SQL). SQLJ enables developers to use Java™ data types or classes as data types in SQL. Therefore, relational tables accessed within an SQLJ application may contain columns having Java™ data types or classes. A SQLJ stored procedure is considered a specialized type of SQLJ application. Additionally, a SQLJ iterator describes columns for a result set using Java™ types.

On the other hand, some systems do not provide such techniques for obtaining the types of a result set. The DB2® Version 5 of the OS/390® platform, on the other hand, does not have a DESCRIBE command. However, a developer can alter the Data Manipulation Language (DML) statement so that it returns no data, but allows full access to the metadata similar to that provided in the above DESCRIBE command.

The types of a result set is useful and desirable information. There is a need in the art of an improved technique of obtaining this information, across platforms.

SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for a computer-implemented technique for cross-platform subselect metadata extraction.

In accordance with the present invention, a metadata extraction system executes a query against a database on a data storage device connected to a computer. Initially, the query is modified to replace one or more selected clauses with a false clause. The modified query with the false clause is executed. Metadata is retrieved from the result set obtained by executing the modified query. The metadata is used to obtain column types, which are converted to Java™ types. Then, a SQLJ iterator is generated, which has parameters for the Java™ types.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 schematically illustrates a hardware environment of an embodiment of the present invention;

FIG. 2 is a hardware environment used to implement a client computer in one embodiment of the invention;

FIG. 3 is a hardware environment used to implement a server computer in one embodiment of the invention;

FIG. 4 illustrates the Stored Procedure Builder topology;

FIG. 5 is a flow diagram of a general overview for building and compiling stored procedures using the present invention; and

FIG. 6 is a flow diagram of the processing performed by the Metadata Extraction System.

DETAILED DESCRIPTION

In the following description of one embodiment of the invention, reference is made to the accompanying drawings which form a part hereof, and which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized as structural changes may be made without departing from the scope of the present invention.

Hardware Environment

FIG. 1 schematically illustrates a hardware environment of an embodiment of the present invention, and more particularly, illustrates a typical distributed computer system using the network 100 to connect client computers 102 executing client applications to a server computer 104 executing software and other computer programs, and to connect the server system 104 to data sources 106.

A typical combination of resources may include client computers 102 that are personal computers or workstations, and a server computer 104 that is a personal computer, workstation, minicomputer, or mainframe. These systems are coupled to one another by various networks, including LANs, WANs, and the Internet. The data sources 106 may be geographically distributed.

A client computer 102 typically executes a client application and is coupled to a server computer 104 executing server software. The client application program is typically a software program which can include, *inter alia*, multi-media based applications, e-mail applications, e-business applications, or workflow applications. The server computer 104 also uses a data source interface and, possibly, other computer programs, for connecting to the data sources 106. The client computer 102 is bi-directionally coupled with the server computer 104 over a line or via a wireless system. In turn, the server computer 104 is bi-directionally coupled with data sources 106. The computer 110 is bidirectionally coupled with the client computers 102 and the server computers 104.

The computer programs executing at each of the computers, including the Secure Access System 110, are comprised of instructions which, when read and executed by the computers, cause the computers to perform the steps necessary to implement and/or use the present invention. Generally, computer programs are tangibly embodied in and/or readable from a device, carrier, or media, such as memory, data storage devices, and/or data communications devices. Under control of an operating system, the computer programs may be loaded from the memory, data storage devices, and/or data communications devices into the memory of each computer for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, system, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media, including the internet. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 1 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention. For example, the system described can apply either to a general-purpose computer, or to a special-purpose system.

FIG. 2 is a hardware environment used to implement a client computer in one embodiment of the invention. The present invention is typically implemented using a client computer 200, which generally includes a processor 202, random access memory (RAM) 204, data storage devices 206 (e.g., hard, floppy, and/or CD-ROM disk drives, etc.), data communications devices 208 (e.g., modems, network interfaces, etc.), display devices 210 (e.g., CRT, LCD display, etc.), and input devices 212 (e.g., mouse pointing device, keyboard, and CD-ROM drive). It is envisioned that attached to the client computer 200 may be other devices, such as read only memory (ROM), a video card, bus interface, printers, etc. Those skilled in the art will recognize that any combination of the above components, or any number of different components, peripherals, and other devices, may be used with the client computer 200.

The client computer 200 operates under the control of an operating system (OS) 214. The operating system 214 is booted into the memory 204 of the client computer 200 for execution when the client computer 200 is powered-on or reset. In turn, the operating system 214 then controls the execution of one or more computer programs 216, such as a Stored Procedure Builder 218, by the client computer 200. The Stored Procedure Builder 218 comprises a Metadata Extraction System 220. The present invention is generally implemented in these computer programs 216, which execute under the control of the operating system 214 and cause the client computer 200 to perform the desired functions as described herein.

The operating system 214 and computer programs 216 are comprised of instructions which, when read and executed by the client computer 200, causes the client computer 200 to perform the steps necessary to implement and/or use the present invention. Generally, the

operating system 214 and/or computer programs 216 are tangibly embodied in and/or readable from a device, carrier, or media, such as memory 204, data storage devices 206, and/or data communications devices 208. Under control of the operating system 214, the computer programs 216 may be loaded from the memory 204, data storage devices 206, and/or data communications devices 208 into the memory 204 of the client computer 200 for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 2 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention.

FIG. 3 is a hardware environment used to implement a server computer in one embodiment of the invention. In the environment, a computer system 302 is comprised of one or more processors connected to one or more data storage devices 304 and 306 that store one or more relational databases, such as a fixed or hard disk drive, a floppy disk drive, a CDROM drive, a tape drive, or other device.

Operators of the computer system 302 use a standard operator interface 308 to transmit electrical signals to and from the computer system 302 that represent commands for performing various search and retrieval functions, termed queries, against the databases.

In one embodiment of the present invention, the RDBMS software comprises the DB2® product for the OS/390® operating systems. Those skilled in the art will recognize,

however, that the present invention has application to any RDBMS software, whether or not the RDBMS software uses SQL.

As illustrated in FIG. 3, the DB2® system for the OS/390® operating system includes three major components: the Internal Resource Lock Manager (IRLM) 310, the Systems Services module 312, and the Database Services module 314. The IRLM 310 handles locking services for the DB2® system, which treats data as a shared resource, thereby allowing any number of users to access the same data simultaneously. Thus concurrency control is required to isolate users and to maintain data integrity. The Systems Services module 312 controls the overall DB2® execution environment, including managing log data sets 306, gathering statistics, handling startup and shutdown, and providing management support.

At the center of the DB2® system is the Database Services module 314. The Database Services module 314 contains several submodules, including the Relational Database System (RDS) 316, the Data Manager 318, the Buffer Manager 320, and other components 322 such as a SQL compiler/interpreter. These submodules support the functions of the SQL language, i.e. definition, access control, interpretation, compilation, database retrieval, and update of user and system data.

The present invention is generally implemented using SQL statements executed under the control of the Database Services module 314. The Database Services module 314 retrieves or receives the SQL statements, wherein the SQL statements are generally stored in a text file on the data storage devices 304 and 306 or are interactively entered into the computer system 302 by an operator sitting at a monitor 326 via operator interface 308. The Database Services module 314 then derives or synthesizes instructions from the SQL statements for execution by the computer system 302.

Generally, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all tangibly embodied in a computer-readable medium, e.g. one or more of the data storage devices 304 and 306. Moreover, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all comprised of instructions which, when read and

executed by the computer system 302, causes the computer system 302 to perform the steps necessary to implement and/or use the present invention. Under control of an operating system, the RDBMS software, the SQL statements, and the instructions derived therefrom, may be loaded from the data storage devices 304 and 306 into a memory of the computer system 302 for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 3 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention.

Cross-Platform Subselect Metadata Extraction

A. Overview

The present invention provides a Metadata Extraction System 220 that extracts metadata across platforms. In particular, the metadata that is extracted includes data types of columns in a result set. Without knowledge of which platform the Metadata Extraction System 220 is running on, the Metadata Extraction System 220 can determine the metadata of a result set of a SELECT statement without returning or producing any customer data from the statement.

B. Stored Procedure Builder

The Metadata Extraction System 220 is part of a Stored Procedure Builder, which assists developers with the creation of stored procedures. A stored procedure has zero or more parameters. The stored procedure receives parameter values and is executed via SQL statements. Typically, a stored procedure is used to execute a group of SQL statements without user input (other than the initial parameter values).

In one embodiment of the invention, the stored procedure builder is the DB2® Stored Procedure Builder, which is a graphical application that supports the rapid development of stored procedures. The Stored Procedure Builder may be used to create stored procedures, build stored procedures on local and remote servers, modify and rebuild existing stored procedures, and run stored procedures for testing and debugging the execution of installed stored procedures. The Stored Procedure Builder provides a single development environment that supports many different platforms and works in conjunction with many different applications.

FIG. 4 illustrates the Stored Procedure Builder 400 topology. The topology includes development, debugging, and deployment components. As to development components, one of many Development Applications 402 is used to launch the Stored Procedure Builder 400, on one of many different Platforms 404. In particular, the Stored Procedure Builder can be launched as a separate application from the DB2® Universal Database® program group or from any of the following development applications: IBM® VisualAge® for Java™ Version 3.0 or later, Microsoft® Visual C++® Version 5 or later, Microsoft® Visual Basic® Version 5 or later, or IBM® DB2® Control Center. Some platforms include: AIX®, OS/2®, OS/390®, OS/400®, Sun Solaris®, UNIX®, Windows NT®, Windows 95®, Windows 98®, and Windows 2000®.

As to debugger components, a Debugger Daemon 406 and Debugger Client 408 are provided for debugging a stored procedure built with the Stored Procedure Builder 400. As to Deployment components, the Stored Procedure Builder 400 uses a Java™ Database Connectivity (JDBC) application programming interface (API) 410 to communicate with a

Database 412. The Stored Procedure Builder is implemented with Java™ code and all database connections are managed by using a Java™ Database Connectivity (JDBC) application programming interface (API). Furthermore, the Database 412 may reside on one of many Platforms 414 and connected to a Debugger Backend 416. Some platforms include:

5 AIX®, OS/2®, OS/390®, OS/400®, Sun Solaris®, UNIX®, Windows NT®, Windows 95®, Windows 98®, and Windows 2000®.

IBM, DB2, Universal Database (UDB), OS/2, OS/390, OS/400, AIX, and VisualAge are trademarks or registered trademarks of International Business Machines, Corporation in the United States and/or other countries.

10 UNIX is a trademark registered of UNIX Systems Laboratories in the United States and/or other countries.

Microsoft, Windows NT, Windows 95, Windows 98, Windows 2000, Visual C++ , and Visual Basic are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

15 Sun Solaris, JAVA, and JAVA-based marks are trademarks or registered trademarks of Sun Microsystems in the United States and/or other countries.

C. Metadata Extraction

A SQLJ iterator is a command that describes the output of a SELECT statement. This is used to statically bind a statement in DB2®. A technique for supporting the SQLJ iterator across platforms was needed. Thus, in order to support the SQLJ iterator across

20 different platforms, the Metadata Extraction System 220 was developed. One advantage of the Metadata Extraction System 220 is that it supports SQLJ iterators (an important part of SQLJ) across different platforms, instead of just supporting it on one or a few platforms. Some platforms include: AIX®, OS/2®, OS/390®, OS/400®, Sun Solaris®, UNIX®, Windows

25 NT®, Windows 95®, Windows 98®, and Windows 2000®.

The Metadata Extraction System 220 alters the SELECT statement by adding/inserting a false WHERE condition (e.g., WHERE 1 = -1). The inclusion of the false

WHERE condition results in no rows being returned, while metadata about the result set is available. In particular, even though no rows are returned, the MetaData via a Java.sql.ResultSetMetaData field is still available and used to construct the iterator necessary for the generating SQLJ code where a SELECT statement is returning rows of data.

5 Since SQL is a complex language where there are a minimal set of reserved words, one or more false SQL statements are constructed and loaded into a list of statements to run. In one embodiment, the number of false SQL statements will be equivalent to the number of WHERE clauses. For example, if a SQL statement has a WHERE clause, then one false WHERE clause is generated and used to replace the original WHERE clause and anything
10 following it (e.g., a GROUP BY clause). If a SQL statement has two WHERE clauses, then one false WHERE clause is generated and used to replace both of the WHERE clauses and everything following them and a second false WHERE clause is generated and used to replace the second WHERE clause and everything following it (i.e., this leaves the first WHERE clause in the SQL statement. The first SQL statement with a false WHERE clause
15 to run without experiencing a SQL exception is the one used to extract the MetaData.

 The Metadata Extraction System 220 improves performance by removing the ORDER BY clauses, should they exist in the statement. Also, the Metadata Extraction System 220 handles subselect statements (e.g., a WHERE clause within a WHERE clause). In one embodiment, the pseudocode is not applied to SELECT <column name> INTO
20 statements because results are directly loaded into variables.

 Additionally, if the SQL statement requires that the application program pass in parameter values at run time (e.g., the parameters could be used in an expression), default values are used by the Metadata Extraction System 220 when executing a SQL statement with one or more false clauses so that the SQL statement can run properly.

25 FIG. 5 is a flow diagram of a general overview for building and compiling stored procedures using the present invention. Note that FIG. 5 describes processing of one SQL statement for illustration only, and one skilled in the art would recognize that multiple statements may be processed and incorporated into a single stored procedure. Additionally,

although the stored procedure builder, precompiler, and compiler are discussed as separate entities, they may be combined in various manners, resulting in fewer separate entities (e.g., the stored procedure builder and precompiler may be combined).

In block 500, a stored procedure builder receives a SQL statement from a user and determines whether the SQL statement requires a SQLJ iterator. A SQL statement requires an iterator if it is a SELECT statement, other than a SELECT INTO statement (which selects data into one or more variables). In block 502, if an iterator is required, processing continues with block 504, otherwise, processing continues with block 506.

In block 504, the Metadata Extraction System 220 assists the stored procedure builder with building a stored procedure requiring a SQLJ iterator through construction of a SQLJ iterator for the SQL statement and continues to block 508 for precompiling. In block 506, the stored procedure builder builds a stored procedure and continues to block 508 for precompiling. In block 508, a precompiler precompiles the stored procedure. In block 510, a Java™ compiler compiles the precompiled stored procedure.

The following provides pseudocode for the Metadata Extraction System 220:

Is there a WHERE clause in the SQL statement?

N Y

1) | + Establish its position in the statement.

| Is it bounded by whitespace on either side?

N Y

| | + Copy SQL statement up until the WHERE clause, and replace the WHERE clause and everything following it with " WHERE 1 = -1 ". Store this statement into a list where items are stored on a first in first out basis.

| | If there is a WHERE clause after the one used previously, using the new WHERE clause position go to #1.

| | + increment the position by 1, and look for more WHERE clauses.

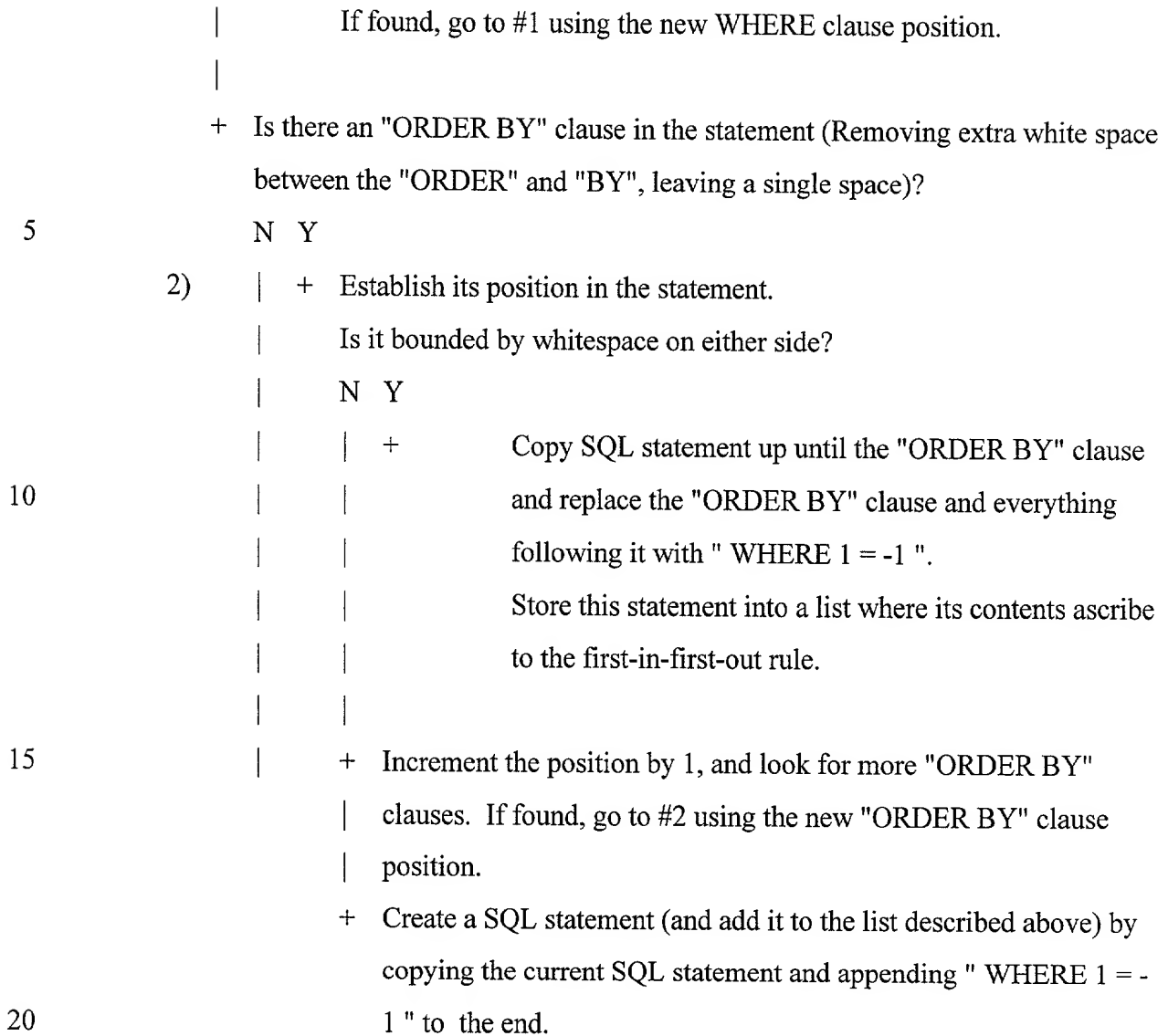


FIG. 6 is a flow diagram of the processing performed by the Metadata Extraction
 System 220. In block 600, the Metadata Extraction System 220 receives a SQL statement.
 In block 602, the Metadata Extraction System 220 processes WHERE and GROUP BY
 clauses in the SQL statement to create a list of SQL statements with false clauses. In block
 25 604, the Metadata Extraction System 220 executes each statement in list in sequence until
 one executes successfully. In block 606, the Metadata Extraction System 220 obtains type
 data for columns from the result set of the successfully executed statement. In block 608,

the Metadata Extraction System 220 converts the type data into JAVA™ types. In block 610, the Metadata Extraction System 220 creates a SQLJ iterator with parameters having the JAVA™ types.

D. Examples

5 D.1 Sample SQLJ code with Different Data Types

In the following example, a default table selection is made via, for example, a user interface on the client computer, with the client and server being on the same machine.

Initially, a user enters an initial SQL statement of: SELECT * FROM SYSCAT.PROCEDURES. Then, the Stored Procedure Builder 218 and Metadata Extraction System 220 work in conjunction to produce SQLJ Stored Procedure QUEST.Procedure4, with a SQLJ iterator that tells the RDBMS how to bind a result set. In particular, the Metadata Extraction System 220 generates the SQLJ iterator. Each parameter of the SQLJ iterator is a Java™ type that corresponds to a type in the initial SQL statement. The Stored Procedure Builder generates the import statements and the public class Procedure4 code, which includes the initial SQL statement and a result set parameter "rs" that returns a result set to a calling application.

```
/**
 * SQLJ Stored Procedure QUEST.Procedure4
 */
20 import java.sql.*;          // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator Procedure4_Cursor1 ( String, String, String, int, String,
                                   short, byte[], String, java.sql.Timestamp,
```

```
String, String, String, String, String, String,
String, String, String, String, String, short,
String, int, String, String );
```

```
public class Procedure4
5  {
    public static void procedure4 ( ResultSet[] rs ) throws SQLException, Exception
    {
        Procedure4_Cursor1 cursor1 = null;
        #sql cursor1 =
10    {
        SELECT * FROM SYSCAT.PROCEDURES
        };

        rs [0] = cursor1.getResultSet ( );
    }
15 }
```

D.2 Sample SQLJ code with Different Data Types

In the following example, a table with different datatypes is accessed. Again the stored procedure is built after receiving user input.

Initially, a user enters a SELECT statement. Then, the Stored Procedure Builder 218 and Metadata Extraction System 220 work in conjunction to produce SQLJ Stored Procedure QUEST.Procedure5, with a SQLJ iterator that tells the RDBMS how to bind a result set. In particular, the Metadata Extraction System 220 generates the SQLJ iterator. Each parameter of the SQLJ iterator is a Java™ type that corresponds to a type in the initial SQL statement. The Stored Procedure Builder generates the import statements and the public class

25 Procedure5 code, which includes the initial SELECT statement and a result set parameter "rs" that returns a result set to a calling application.

```

/**
 * SQLJ Stored Procedure QUEST.Procedure5
 */
import java.sql.*;          // JDBC classes
5  import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator Procedure5_Cursor1 ( String, String, String, java.math.BigDecimal,
                                   short, int, long, float, double, java.sql.Date,
                                   Java.sql.Time, java.sql.Timestamp );

10 public class Procedure5
    {
        public static void procedure5 ( ResultSet[ ] rs ) throws SQL Exception, Exception
        {
            Procedure5_Cursor1 cursor1 = null;
15         #sql cursor1 =
            {
                SELECT
                    ALLTYPES.CHAR AS CHAR,
                    ALLTYPES.VARCHAR AS VARCHAR,
20         ALLTYPES.LONGVARCHAR AS LONGVARCHAR,
                    ALLTYPES.DECIMAL AS DECIMAL,
                    ALLTYPES.SMALLINT AS SMALLINT,
                    ALLTYPES.INTEGER AS INTEGER,
                    ALLTYPES.BIGINT AS BIGINT,
25         ALLTYPES.REAL AS REAL,
                    ALLTYPES.DOUBLE AS DOUBLE,

```

```

ALLTYPES.DATE AS DATE,
ALLTYPES.TIME AS TIME,
ALLTYPES.TIMESTAMP AS TIMESTAMP
FROM
5      ALLTYPES
      }
      rs[0] = cursor1.getResultSet ( );
    }
  }

```

10

Once metadata is obtained by executing a SQL statement with a false clause, JDBC to Java™ conversion is performed. In one embodiment, a hash code conversion table is used. A portion of the hash code conversion table used for the conversion that can be seen in Example 2, in which the ALLTYPES table is accessed, follows:

```

15      jdbcJava.put (new Integer(Types.CHAR), "String");
      jdbcJava.put (new Integer(Types.VARCHAR), "String");
      jdbcJava.put (new Integer(Types.LONGVARCHAR), "String");
      jdbcJava.put (new Integer(Types.NUMERIC), "java.math.BigDecimal");
      jdbcJava.put (new Integer(Types.DECIMAL), "java.math.BigDecimal");
20      jdbcJava.put (new Integer(Types.BIT), "boolean");
      jdbcJava.put (new Integer(Types.TINYINT), "byte");
      jdbcJava.put (new Integer(Types.SMALLINT), "short");
      jdbcJava.put (new Integer(Types.INTEGER), "int");
      jdbcJava.put (new Integer(Types.BIGINT), "long");
25      jdbcJava.put (new Integer(Types.REAL), "float");
      jdbcJava.put (new Integer(Types.FLOAT), "double");
      jdbcJava.put (new Integer(Types.DOUBLE), "double");

```

```

jdbcJava.put (new Integer(Types.BINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.VARBINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.LONGVARBINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.DATE), "java.sql.Date");
5 jdbcJava.put (new Integer(Types.TIME), "java.sql.Time");
jdbcJava.put (new Integer(Types.TIMESTAMP), "java.sql.Timestamp");

```

Each of the Types.xxxx are defined constants in the class Types that is in the package java.sql. The class Types defines constants that are used to identify generic SQL types, called JDBC types.

10 D.3 Sample SQLJ code with Different Data Types

In the following example, a sample SQL statement is used that includes WHERE clauses.

The following tables AB and XY illustrate sample tables before a stored procedure is executed against them. All columns are declared as INTEGER types in the example.

15 The following statement is used to display the contents of table TBLAB: db2 =>
select * from tblab.

TABLE AB	COLA	COLB
	-----	-----
20	1	12
	2	12
	3	13
	4	14

4 record(s) selected.

The following statement is used to display the contents of table TBLXY: db2 =>
select * from tblxy

TABLE XY	COLX	COLY
	-----	-----
5	2	22
	3	23

2 record(s) selected.

The following stored procedure, SQLJ Stored Procedure QUEST.Procedure6, is built with the Stored Procedure Builder 218 in conjunction with the Metadata Extraction System 220 and includes a SQLJ iterator that tells the RDBMS how to bind a result set.

Initially, a user enters the following statement: SELECT COLA FROM TBLAB WHERE COLA > ALL (SELECT COLX FROM TBLXY WHERE COLX<0). Note that the SELECT COLX statement within the SELECT COLA statement is referred to as a subselect statement. Also, the SELECT COLA statement has two WHERE clauses. Thus, the Metadata Extraction System 220 generates two different SQL statements with false clauses.

```

/**
 * SQLJ Stored Procedure QUEST.Procedure6
 */
20 import java.sql.*;          // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

```

```
#sql iterator Procdure6_Cursor1 ( int );
```

```
public class Procedure6
```

```
{
```

```
    public static void procedure6 ( ResultSet[] rs ) throws SQLException, Exception
```

```
5      {
```

```
        Procedure6_Cursor1 cursor1 = null;
```

```
        #sql cursor1 =
```

```
        {
```

```
            SELECT COLA FROM TBLAB
```

```
10
```

```
            WHERE COLA > ALL (SELECT COLX FROM TBLXY
```

```
                WHERE COLX<0 )
```

```
        };
```

```
        rs[0] = cursor1.getResultSet();
```

```
    }
```

```
15 }
```

The following table represents the result of executing the stored procedure public class Procedure6:

```
20
```

COLA

1

2

3

```
25
```

4

4 record(s) selected.

Conclusion

This concludes the description of one embodiment of the invention. The following describes some alternative embodiments for accomplishing the present invention. For example, any type of computer, such as a mainframe, minicomputer, or personal computer, or
5 computer configuration, such as the Internet, a local area network, or wide area network, could be used with the present invention.

The foregoing description of one embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in
10 light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.

WHAT IS CLAIMED IS:

- 1 1. A method for executing a query against a database on a data storage device
2 connected to a computer, the method comprising:
3 modifying the query to replace one or more selected clauses with a false clause;
4 executing the modified query with the false clause; and
5 retrieving metadata from the result set obtained by executing the modified query.
- 1 2. The method of claim 1, wherein the query comprises a SELECT statement.
- 1 3. The method of claim 2, wherein the SELECT statement is not a SELECT
2 INTO statement.
- 1 4. The method of claim 1, wherein the selected clauses comprise WHERE
2 clauses.
- 1 5. The method of claim 1, wherein the selected clauses comprise GROUP BY
2 clauses.
- 1 6. The method of claim 1, wherein modifying the query comprises:
2 generating a list of modified queries, wherein each modified query has one or more
3 selected clauses replaced with a false clause; and
4 executing each modified query until one executes successfully.
- 1 7. The method of claim 6, wherein a query executes successfully if it executes
2 without an exception.

1 8. The method of claim 1, wherein the metadata comprises column type data for
2 the result set.

1 9. The method of claim 8, further comprising converting the column type data to
2 JAVA types.

1 10. The method of claim 9, further comprising generating a SQLJ iterator with
2 parameters having the JAVA types.

1 11. The method of claim 1, further comprising determining whether the query
2 requires a SQLJ iterator.

1 12. An apparatus for executing a query, comprising:
2 a computer connected a data storage device that stores a database containing data;
3 one or more computer programs, performed by the computer, for modifying the query
4 to replace one or more selected clauses with a false clause, executing the modified query with
5 the false clause, and retrieving metadata from the result set obtained by executing the
6 modified query.

1 13. The apparatus of claim 12, wherein the query comprises a SELECT statement.

1 14. The apparatus of claim 13, wherein the SELECT statement is not a SELECT
2 INTO statement.

1 15. The apparatus of claim 12, wherein the selected clauses comprise WHERE
2 clauses.

1 16. The apparatus of claim 12, wherein the selected clauses comprise GROUP BY
2 clauses.

1 17. The apparatus of claim 12, wherein modifying the query comprises:
2 generating a list of modified queries, wherein each modified query has one or more
3 selected clauses replaced with a false clause; and
4 executing each modified query until one executes successfully.

1 18. The apparatus of claim 17, wherein a query executes successfully if it executes
2 without an exception.

1 19. The apparatus of claim 12, wherein the metadata comprises column type data
2 for the result set.

1 20. The apparatus of claim 19, further comprising converting the column type data
2 to JAVA types.

1 21. The apparatus of claim 20, further comprising generating a SQLJ iterator with
2 parameters having the JAVA types.

1 22. The apparatus of claim 12, further comprising determining whether the query
2 requires a SQLJ iterator.

1 23. An article of manufacture comprising a computer program carrier readable by
2 computers and embodying one or more instructions executable by the computer for executing
3 a query against a database on a data storage device connected to the computer, comprising:
4 modifying the query to replace one or more selected clauses with a false clause;
5 executing the modified query with the false clause; and
6 retrieving metadata from the result set obtained by executing the modified query.

1 24. The article of manufacture of claim 23, wherein the query comprises a
2 SELECT statement.

1 25. The article of manufacture of claim 24, wherein the SELECT statement is not
2 a SELECT INTO statement.

1 26. The article of manufacture of claim 23, wherein the selected clauses comprise
2 WHERE clauses.

1 27. The article of manufacture of claim 23, wherein the selected clauses comprise
2 GROUP BY clauses.

1 28. The article of manufacture of claim 23, wherein modifying the query
2 comprises:
3 generating a list of modified queries, wherein each modified query has one or more
4 selected clauses replaced with a false clause; and
5 executing each modified query until one executes successfully.

1 29. The article of manufacture of claim 28, wherein a query executes successfully
2 if it executes without an exception.

[illegible][illegible][illegible]

- [illegible]

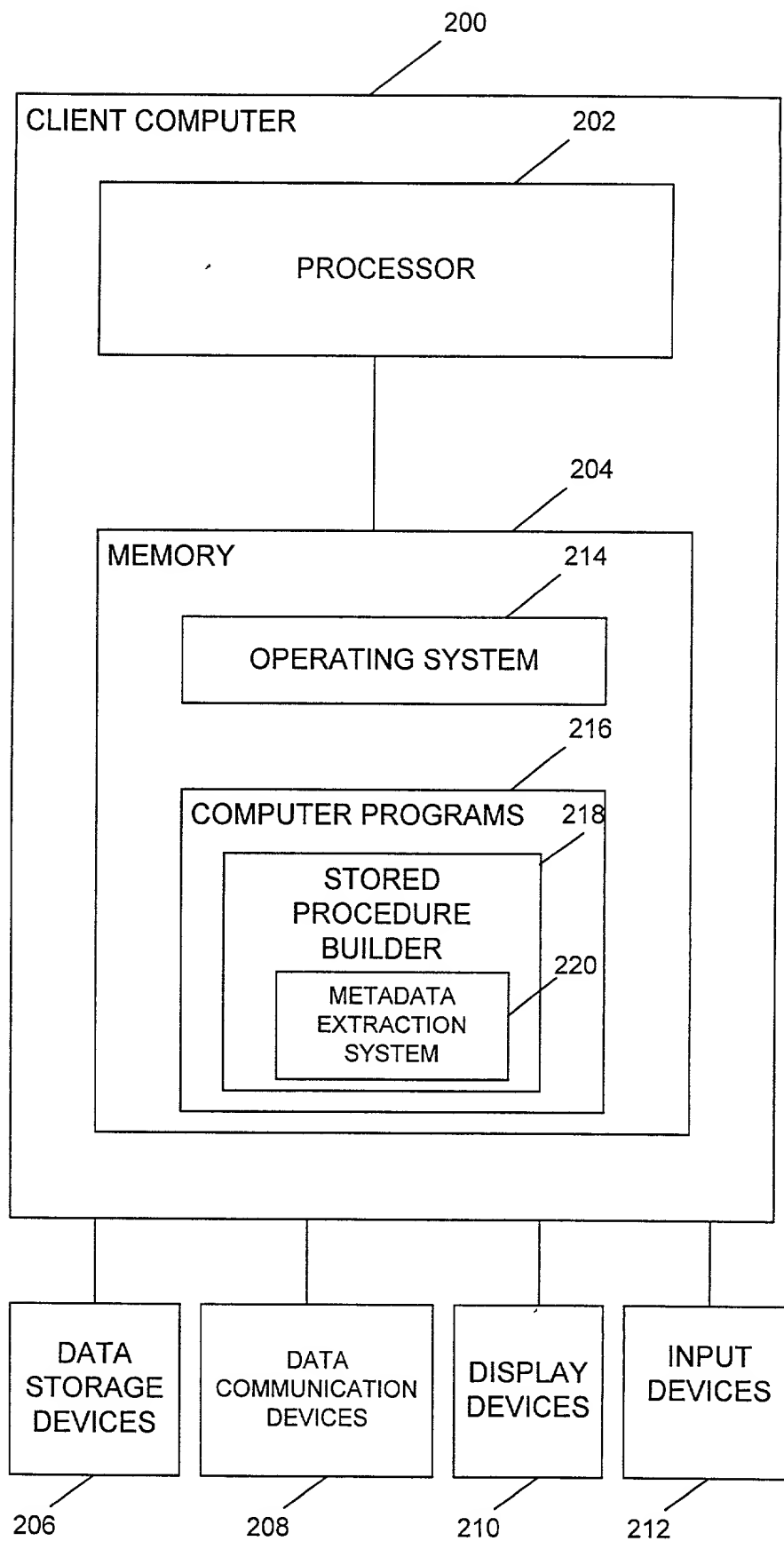


FIG. 2

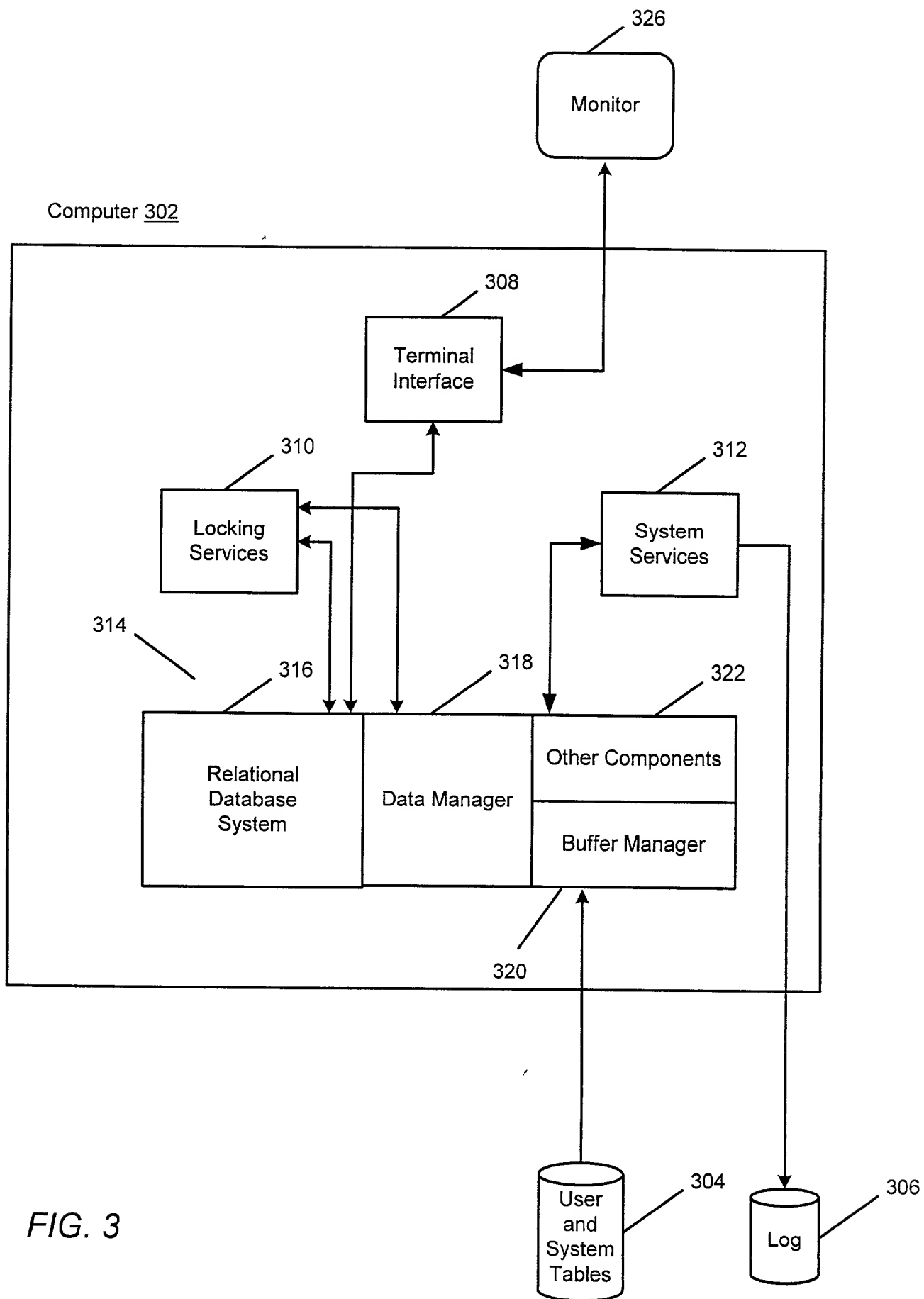


FIG. 3

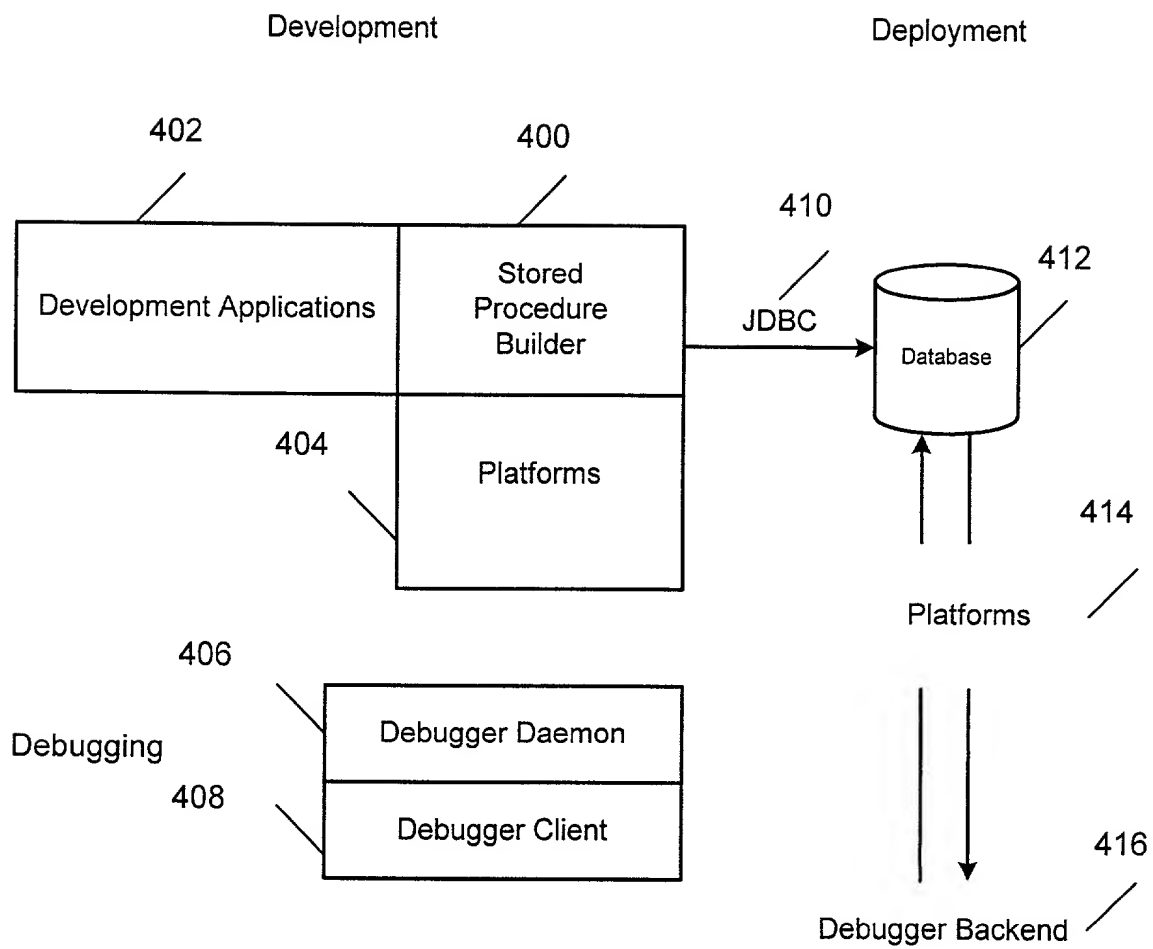


FIG. 4

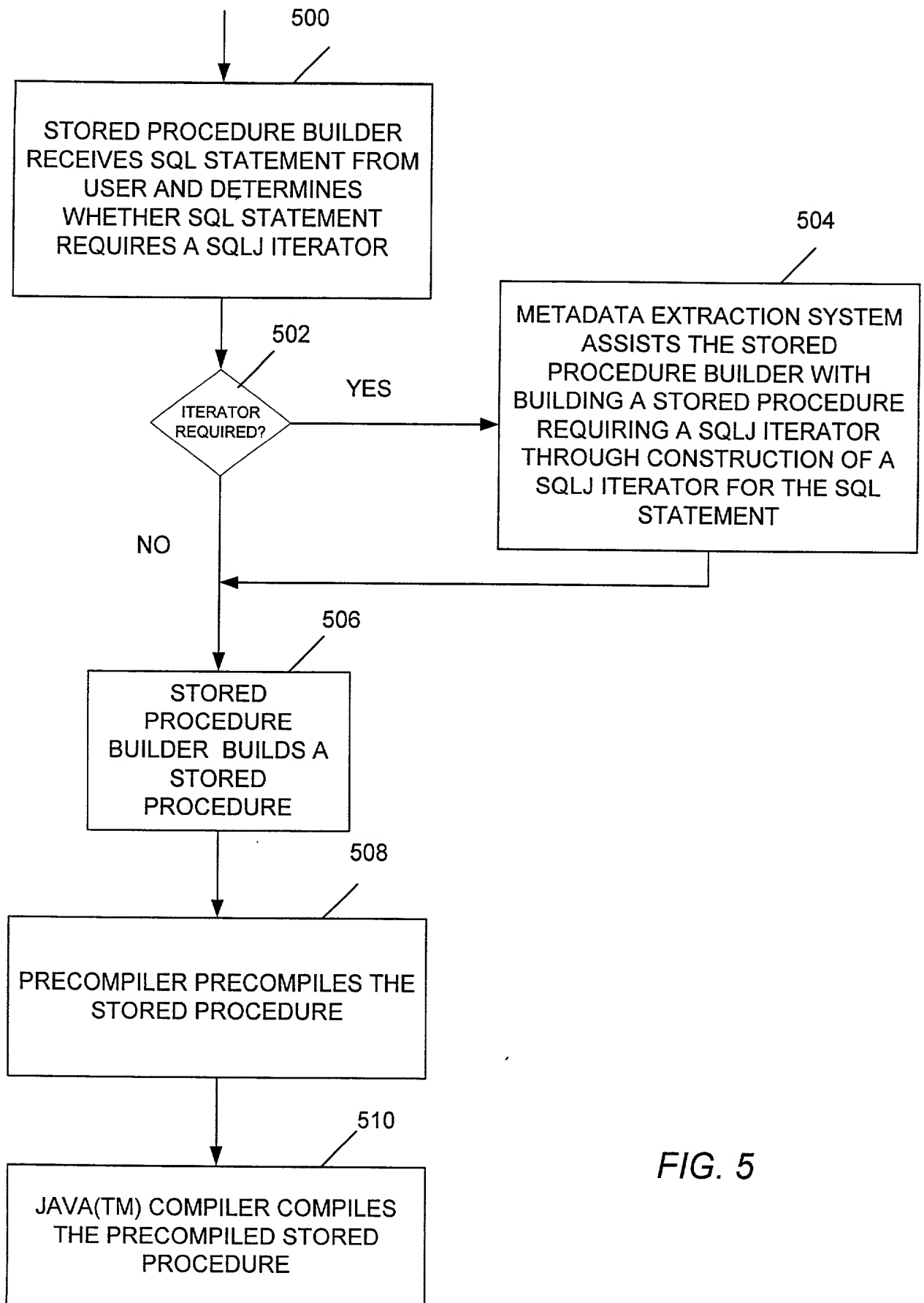


FIG. 5

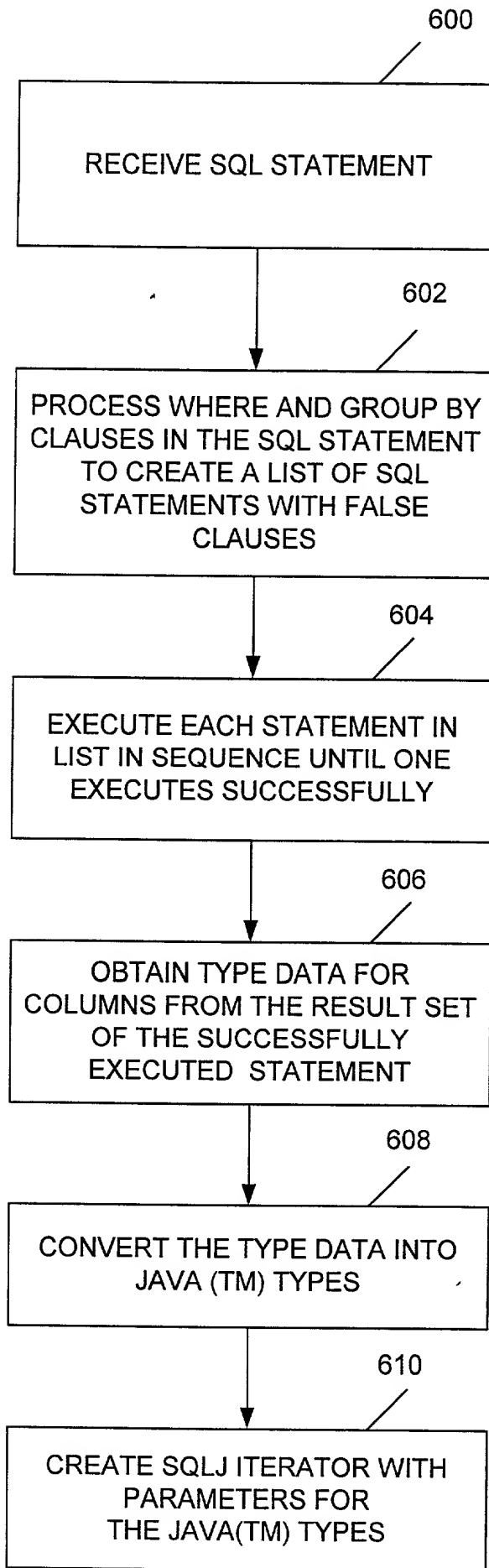


FIG. 6

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999179

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

CROSS-PLATFORM SUBSELECT METADATA EXTRACTION

the specification of which (check one)

X is attached hereto.

_____ was filed on _____

as Application Serial No. _____

and was amended on _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
_____	_____	_____	____ Yes	____ No
(Number)	(Country)	(Day/Month/Year Filed)		

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56, which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

_____	_____	_____
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

DOCKET: ST999179

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. (list name and registration number)

Romualdas Strimaitis, Registration No. 35,697, Prentiss W. Johnson, Registration No. 33,123, Ingrid M. Foerster, Registration No. 36,511, Timothy M. Farrell, Registration No. 37,321, Christopher A. Hughes, Registration No. 26,914, John E. Hoel, Registration No. 26,279, Edward A. Pennington, Registration No. 32,588, and Joseph C. Redmond, Jr., Registration No. 18,753, all of whom are attorneys with IBM Corporation; and

Edward G. Poplawski, Registration No. 33,439, Laurence H. Pretty, Registration No. 25,312, Robert A. Schroeder, Registration No. 25,373, Janaki Komanduri, Registration No. 40,684, Denise L. McKenzie, Registration No. 43,790, Gregory S. Cordrey, Registration No. 44,089, and Matthew C. Lapple, Registration No. 44,203, all of whom are attorneys with the law firm of Pretty, Schroeder & Poplawski.

Send correspondence to: Janaki Komanduri, Esq.
Pretty, Schroeder & Poplawski
444 S. Flower Street, 19th Floor
Los Angeles, California 90071

Direct Telephone Calls to: (name and telephone number) Janaki Komanduri, (213) 622-7700

Full name of sole or first joint-inventor: Richard Henry Mandel, III

Inventor's signature:

Richard Henry Mandel III

Date: 4.4.2000

Residence: 6275 Shadelands Dr., San Jose, California 95123

Citizenship: U.S.A.

Post Office Address: Same